Sun Reference No.: P4518
Our Reference No.: 06502.0302

UNITED STATES. PATENT APPLICATION

of

PAUL J. HINKER

for

AUTOMATIC CONVERSION OF SOURCE
CODE FROM 32-BIT TO 64-BIT

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

FIELD OF THE INVENTION

The present invention relates generally to data processing systems and, more particularly, to the automatic generation of interfaces to convert 32-bit code to 64-bit code.

BACKGROUND OF THE INVENTION

The physical memory of a computer, i.e., random access memory ("RAM"), consists of a number of cells which store information. These cells are referred to as addresses. Programs access the memory by referring to an address space. If a memory cell consists of N bits, it can hold 2^N different bit combinations. Thus, if a memory cell consists of 32 bits, it can hold 2^{32} different bit combinations. A program written for a 32-bit memory addressing scheme may access 2^{32} memory addresses, the equivalent of four gigabytes of RAM.

Most conventional programs follow a 32-bit addressing model. Thus, if a computer has more than four gigabytes of RAM, the processor cannot directly address all of the physical memory without using complicated memory access schemes. The same access problems may occur when accessing files maintained in a secondary storage device, e.g., a database maintained on a hard disk, which is larger than four gigabytes.

Programs originally written according to a 32-bit addressing model are unable to make calls to, or directly address, a larger address space without rewriting the source code, a time consuming and daunting task, or using complex addressing schemes. Rewriting the source code is possible only if the original source code is available.

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

Accordingly, a need exists for a manner of adapting source code written for a 32-bit addressing model to execute on machines having an address space larger than four gigabytes.

SUMMARY OF THE INVENTION

In accordance with methods and systems consistent with the present invention, a system that automatically generates 64-bit interfaces to programs written according to a 32-bit addressing model is provided. These interfaces are automatically generated to allow backward compatibility with programs that assume 32-bit addressing.

In accordance with an implementation of methods consistent with the present invention, a method is provided in a data processing system that receives 32-bit source code and automatically generates an interface between 32-bit source code and 64-bit library routines.

In accordance with another implementation, a method is provided in a data processing system having source code with a subprogram having at least one of an integer and logical parameter. The method reads the source code and generates a stub routine that invokes the subprogram and that converts 32-bit integer/logical parameters to 64-bit parameters and calls corresponding 64-bit library routines.

In accordance with an implementation of systems consistent with the present invention, a computer-readable memory device encoded with a program having instructions for execution by a processor is provided. The program comprises source code of a subprogram with a parameter. The program also comprises a stub routine that

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

receives a set of parameter values and creates the values for the required parameters from the received set of parameter values to invoke the subprogram, where the received set of parameter values contains at least one of an integer and logical parameter.

In another implementation of systems consistent with the present invention, a data processing system is provided. This data processing system contains a storage device and a processor. The storage device comprises source code of a subprogram having a parameter and an interface generator that reads the subprogram and that generates an interface file with indications of characteristics of the parameter. The storage device also comprises a stub generator that reads the interface file and that generates a stub routine that converts integer and logical parameters from 32-bit to 64-bit. Each of the stubs receives a set of parameter values, generates the parameter from the received set of parameter values, and invokes the subprogram with the values for the parameter. The processor runs the interface generator and the stub generator.

BRIEF DESCRIPTION OF THE DRAWINGS

This invention is pointed out with particularity in the appended claims. The above and further advantages of this invention may be better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

Figure 1 depicts a data processing system suitable for use with methods and systems consistent with the present invention;

Figure 2 depicts a flow chart of the steps performed to automatically generate 64-bit interfaces in accordance with methods and systems consistent with the present invention; and

Figures 3A and 3B depict a flow chart of the steps performed by the interface generator depicted in Figure 2.

Figures 4A and 4B depict a flow chart of the steps performed by the stub generator.

DETAILED DESCRIPTION

In accordance with methods and systems operating in accordance with the principles of the present invention, an automatic method of generating 32 to 64 bit conversion stubs is provided. This method allows callers to invoke a 32-bit interface to call the underlying 64-bit code, thus maintaining backward compatibility for pre-existing 32-bit code without having to rewrite the 32-bit code.

Overview

Methods and systems operating in a manner that is consistent with the present invention provide a script that scans the 32-bit source code and that generates an interface file for each subprogram. This file defines the signature for the associated subprogram, including its name, its parameters, and each parameter's type. This script then scans the 32-bit source code again and inserts code-generator statements into each interface file. These code-generator statements provide meaningful information about the integer parameters to facilitate the automatic generation of 64-bit interfaces. After the code-generator statements are added, another script is run that reads each interface file and

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

automatically generates a number of stub routines, which serve as the 32-bit interfaces to the 64-bit subprogram.

Implementation Details

Figure 1 depicts a data processing system 100 suitable for use with methods and systems consistent with the present. Data processing system 100 includes a memory 102, a secondary storage device 104, an input device 106, a central processing unit (CPU) 108, and a video display 110. In the memory 102 resides an interface generator 111 and a stub generator 112. Interface generator 111 reads 32-bit source code 114 in secondary storage 104, and generates 32-bit interfaces 116, one for each subprogram encountered in the source code. Stub generator 112 reads 32-bit interface files 116 and generates 32 to 64 bit conversion stubs 118 so that 32-bit code 114 can utilize the 32 to 64 bit conversion stubs 118 to invoke the 64-bit code 122.

Following is the definition of the 32-bit interface file, where the words INTERFACE, SUBROUTINE, FUNCTION, and END are keywords and the word TYPE represents any valid Fortran type (i.e., INTEGER, LOGICAL, REAL, CHARACTER, or COMPLEX):

Table 1

```
INTERFACE  Interface_Name
     {SUBROUTINE  |  TYPE  FUNCTION}  (Parameter1,
     [Parameter2, . . . , ParameterN])
     TYPE Parameter1
     TYPE Parameter2
     . . .
     TYPE ParameterN
  END SUBROUTINE
END INTERFACE
```

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

Following is an example of a 32-bit interface file for the CAXPY Fortran 77 subprogram, which performs the addition of two vectors X and Y and adds a constant Alpha:

Table 2

```
INTERFACE CAXPY
        SUBROUTINE CAXPY (N, ALPHA, X, INCX, Y, INCY)
        INTEGER    : : N
        COMPLEX  : : ALPHA
        COMPLEX  : : X (*)
        INTEGER    : : INCX
        COMPLEX  : : Y (*)
        INTEGER    : : INCY
  END SUBROUTINE
END INTERFACE
```

Figure 2 depicts a flowchart of the steps performed by methods and systems consistent with the present invention when creating the stubs. The first step performed is to invoke the interface to generate the 32-bit interface file from the 32-bit code (step 202). In this step, the interface generator scans the 32-bit source code and creates an interface file for each subprogram contained in it according to the definition provided above. The interface generator then adds code-generator statements to the interface file. It parses the arguments of each subprogram and adds a comment line that provides meaningful information so that the stub generator can generate a stub. For example, such meaningful information may include a type and shape of a given parameter. After invoking the interface generator, the user invokes the stub generator (step 204). The stub generator reads the interface files and generates stub routines by using the code-generator statements. The stub generator also produces interfaces for the stub routines. These interfaces are used to resolve references during compilation of the 64-bit program. Once

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

generated, the stubs are compiled and can be linked into the 32-bit source code to enable their invocation from the 32-bit source code (step 206).

Figures 3A and 3B depict a flowchart of the steps performed by the interface generator. The first step performed by the interface generator is to select a subprogram from the 32-bit code (step 302). Next, the interface generator creates an interface file for this subprogram (step 304). In this step, the interface generator generates a definition for the subprogram similar to that described above with respect to Table 1. After creating the interface file, the interface generator determines whether there are more subprograms in the 32-bit code (step 306). If so, processing continues to step 302 to create additional interface files for each subprogram.

Otherwise, the interface generator performs a second pass through the 32-bit code by selecting a subprogram and its corresponding interface file (step 308). Next, the interface generator selects a parameter within the subprogram (step 316).

The arguments for the subprograms in the 32-bit code contain comments that provide a significant amount of information about that parameter, such as whether the parameter is an input, output, or input/output parameter; its type; and the meaning associated with its values. In accordance with methods and systems consistent with the present invention, the parameter descriptions closely conform to the following form:

Table 3

Parameter Name	Comment Line in Source Code
N	(input) INTEGER The order of the matrix A. N >= 0.

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

D (input/output) COMPLEX array, dimension (N)
On entry, the diagonal elements of A.
On exit, the diagonal elements DD.

L (input/output) COMPLEX array, dimension (N-1)
On entry, the subdiagonal elements of A.
On exit, the subdiagonal elements of LL and DD.

SUBL (output) COMPLEX array, dimension (N-2)
On exit, the second subdiagonal elements of LL.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of matrix B. NRHS >= 0.

B (input/output) COMPLEX array, dimension (LDB, NRHS)
On entry, the N-by-NRHS right hand side matrix B.
On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB >= max (1, N).

IPIV (input) INTEGER array, dimension (N)
Details of the interchanges and block pivot. If IPIV (K) > 0, 1 by 1 pivot, and if IPIV (K) = K + 1 an interchange done; If IPIV (K) < 0, 2 by 2 pivot, no interchange required.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -k, the k-th argument had an illegal value
> 0: if INFO = k, D (k) is exactly zero. The factorization has been completed, but the block diagonal matrix DD (that is D (K)) is exactly singular, and division by zero will occur if it is used to solve a system of equations.

INPUT The value of an intent(input) parameter is expected to be set by the calling routine. This value may or may not be changed in the callee routine but any changes are not propagated back to the caller.

OUTPUT The value of an intent(output) parameter is undefined when entering the callee routine. This parameter's value is expected to be set by the callee routine and passed back to the caller.

INOUT The value of an intent(inout) parameter is expected to be set by the calling routine. This value may or may not be changed in the callee routine and any changes are propagated back to the caller.

Next, the interface generator inserts the directionality or intent of the parameter into the interface file (step 334). In this step, the interface generator determines if the parameter is an input/output, input, or output parameter by examining the comments in the source code. After making this determination, either an input/output, input, or output code-generator statement is inserted into the interface file. The parameters to be considered are integer or logical parameters as all other parameter types are passed directly through the 32 to 64 bit conversion stub.

If the parameter is an input/output parameter, it is passed with input and output semantics, as required by the language. In the case of C interfaces, this means that the C interface passes a scalar parameter by reference. This parameter allows the compiler to perform optimizations across subprogram boundaries.

If the parameter is an INPUT parameter, it is passed with input semantics. In the case of C interfaces, this means that the C interface can pass the parameter by value. This parameter allows the compiler to perform optimizations across subprogram boundaries.

If the parameter is an OUTPUT parameter, it is passed with output semantics. In the case of C interfaces, this means that the C interface needs to pass a scalar parameter by reference. This parameter allows the compiler to perform optimizations across subprogram boundaries.

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

After inserting the directionality, the interface generator inserts the size of the parameter along each of its dimensions. After inserting the size of the parameter, an EXTENT-code generator statement describing the size and dimensions of a parameter is inserted into the interface file as shown in Fig. 3B (step 336). Again, the parameters to be considered are either integer or logical non-scalar parameters.

The following is an example of the affect of the EXTENT statement on the generated code:

Table 4

```
INTERFACE PAREXTENT
 SUBROUTINE SUB (M,N,A)
  INTEGER :: M !#INPUT
  INTEGER :: N !#INPUT
  INTEGER, DIMENSION(:,:) :: A !#EXTENT(M,N),#INOUT
 END SUBROUTINE
END INTERFACE
```

The above interface description generates the following 32 to 64 bit conversion stub when passed to the stub generator:

Table 5

```
SUBROUTINE SUB(M,N,A)
IMPLICIT NONE

INTEGER*4 M
INTEGER(8) :: M_64
INTEGER*4 N
INTEGER(8) :: N_64
INTEGER*4 M
INTEGER*4 A(M,*)
INTEGER(8),DIMENSION(:,:),ALLOCATABLE :: A_64
INTEGER IA
INTEGER JA
INTEGER*8 MEMSTAT,MAX
INTRINSIC MAX
```

```
15
16 M_64 = M
17 N_64 = N
18 ALLOCATE(A_64(MAX(1,M),MAX(1,N)),STAT=MEMSTAT)
19 IF(MEMSTAT .NE. 0) THEN
20 CALL DSS_MEMERR_64('SUB',MAX(1,M,N)*1_8)
21 STOP
22 ENDIF
23 DO JA = 1, N
24 DO IA = 1, M
25 A_64(IA,JA) = A(IA,JA)
26 END DO
27 END DO
28
29 CALL SUB_64(M_64,N_64,A_64)
30
31 DO JA = 1, N
32 DO IA = 1, M
33 A(IA,JA) = A_64(IA,JA)
34 END DO
35 END DO
36 DEALLOCATE(A_64)
37 RETURN
38 END
```

Lines 1-9 contain declarations. Line 10 declares a two-dimensional integer 64-bit array that can be dynamically allocated. Lines 11-12 declare loop indices. Lines 13-14 declare local variables. Lines 16-17 copy integer parameters with intent(input) to their 64-bit counterparts. Line 18 allocates memory to contain the 32-bit integer, two-dimensional array (A). Lines 19-22 ensure the memory was successfully allocated and report the failure and exit if the allocation was unsuccessful. Lines 23-27 copy the elements of the 32-bit array (A) into the newly allocated 64-bit array (A_64), since the array was declared as intent(inout). Line 29 calls the 64-bit routine. Lines 31-35 copy the elements of the 64-bit array (A_64) into the original 32-bit array (A), since the array was declared as intent(inout). Line 36 deallocates the 64-bit array.

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

Here is another example of the EXTENT (expr[,expr]):

INTEGER,DIMENSION(:) ::A!#EXTENT(N)

In this example, the parameter A is a 1 dimensional array with N elements.

Here is another example of the EXTENT (expr[,expr]):

INTEGER, DIMENSION(:,:) ::B!#EXTENT(2*N,MAX(M,N))

In this example, the parameter B is a 2 dimensional array of 2*N elements in the first dimension, and MAX(M,N) elements in the second dimension.

Next, the interface generator determines when a parameter should be written and describes such condition or conditions (step 340). The NOTOUCH (condition) code-generator statement is used only on parameters having either a LOGICAL or INTEGER type. An example of the NOTOUCH statement follows:

LOGICAL,DIMENSION(:) :: SELECT !#NOTOUCH(HOWMNY.EQ.'A')

In this example, no elements to the 1 dimensional array SELECT should be written by the 32- to 64-bit interface. This is usually necessary when a routine has parameters which are not used and the user may send a "dummy" parameter in the place of an unused parameter.

Another example of a NOTOUCH statement is:

INTEGER :: IL !#NOTOUCH(RANGE.EQ.'A'.OR.RANGE.EQ.'V')

In this example, the scalar, integer parameter IL should not be written by the 32-bit to 64-bit interface when RANGE parameter equals either the character "A" or the character "V".

Next, the interface generator determines if more parameters remain to be processed (step 350), and if so, processing continues to step 316. Otherwise, the interface generator determines if more subprograms remain for processing (step 352), and if so, processing continues to step 308. If no more subprograms remain to be processed, processing ends.

For an example of inserting code-generator statements into an interface file, consider the following. The CSTSV subprogram computes the solution to a complex system of linear equations A * X = B, where A is an N-by-N symmetric tridiagonal matrix and X and B are N-by-NRHS matrices. The following interface file, as shown in Table 6, is generated by examining the CSTSV 32-bit source to extract the parameter list and the parameter declarations.

Table 6

```
INTERFACE
        SUBROUTINE CSTSV (N, NRHS, L, D, SUBL, B, LDB, IPIV,
        INFO)
                INTEGER   : : N
                INTEGER   : : NRHS
                COMPLEX : : L (*)
                COMPLEX : : D (*)
                COMPLEX : : SUBL (*)
                COMPLEX : : B (LDB, *)
                INTEGER   : : LDB
                INTEGER   : : IPIV (*)
                INTEGER   : : INFO
        END SUBROUTINE
END INTERFACE
```

By parsing the comments in the source code, the interface generator can add code-generator statements to the interface file. For instance, the following example line in the 32-bit source code:

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

N (input) INTEGER

allows the interface generator to insert the #INPUT code-generator statement into the interface file associated with the parameter N.

Also, the following exemplary 32-bit source code declarations:

D (input / output) COMPLEX array, dimension (N)

L (input / output) COMPLEX array, dimension (N-1)

SUBL (output) COMPLEX array, dimension (N-2)

NRHS (input) INTEGER

allows the interface generator to not only associate the #INOUT statement with the parameters D and L, but also the #OUTPUT statement can be associated with the SUBL parameter and the #INPUT statement to the NRHS parameter. In addition, the declaration of D gives the interface generator enough information to construct a default value for the parameter N.

Furthermore, the following exemplary 32-bit declaration for B:

B (input / output) COMPLEX array, dimension (LDB, NRHS)

provides enough information to associate the #INOUT statement with B, create a default value for the LDB and NRHS parameters.

This process continues until all the comments have been examined and code-generator statements generated. The final result is an interface file, as shown in Table 7 , populated with code-generator statements.

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

Table 7

```
INTERFACE
      SUBROUTINE CSTSV (N, NRHS, L, D, SUBL, B, LDB, IPIV,
      INFO)
            INTEGER   : : N !#INPUT, #D (#SIZE (D, #DIM=1))
            INTEGER   : : NRHS !#D (#SIZE (B, #DIM=2))
            COMPLEX : : L (*) !#INOUT
            COMPLEX : : D (*) !#INOUT
            COMPLEX : : SUBL (*) !#OUTPUT
            COMPLEX : : B (LDB, *) !#INOUT
            INTEGER   : : LDB  !#D (#STRIDE (B, #DIM=2))
            INTEGER   : : IPIV (*) !#OUTPUT
            INTEGER   : : INFO !#INFO
      END SUBROUTINE
END INTERFACE
```

Figures 4A and 4B depict a flowchart of the steps performed by the stub generator. The stub generator performs two passes through the interface file that has been marked up with the code-generator statements. The first pass discovers information regarding each subprogram and its parameters and begins to populate a hash table with such information. The second pass through each subprogram provides more detailed information to the hash table. Once the hash table has been populated, the stub generator generates stubs using this information. The first step performed by the stub generator is to select a subprogram (step 402). Next, the stub generator determines whether the subprogram is a subroutine (i.e., does not return a return code) or is a function (i.e., returns a return code) (step 404). Next, the stub generator records the name of the subprogram into a hash table, one entry for each subprogram (step 406). Each hash table entry has the following items of information where items 2-14 are specified for each parameter of the subprogram:

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

Table 8

1) Subprogram Name
2) Parameter Name
3) Type (logical, real, double, etc.)
4) Rank (shape)
5) Optional: true/false
6) Info: true/false or expression indicating whether an error has occurred.
7) Work: expression indicating amount of memory needed for this parameter.
8) Sizer: this parameter describes the size of another parameter, the name of that parameter is stored in this field.
9) Mysizer: if another parameter is a sizer for this parameter, that parameter's name is stored in this field.
10) Strider: if this parameter is a strider for another parameter, then its name is stored in this field.
11) Mystrider: if another parameter acts as the strider for this parameter, then its name is stored in this entry.
12) Intent: undefined, input, output, or i/o.

After recording the name, the stub generator examines the parameter list to determine the number of parameters as well as their name for the subprogram and stores this information into the hash table (step 408). The stub generator then identifies the details of each parameter including its shape and type and stores this into the hash table (step 410). After identifying the parameter details, the stub generator determines if there are more subprograms (step 412), and if so, proceeds back to step (402).

Otherwise, the stub generator proceeds to the second pass by selecting a subprogram (step 414). Next, the stub generator processes the code-generator statements by inserting various information into the hash table (step 416). The following table indicates the code-generator statements and the processing that occurs for each one:

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

Table 9

Code-Generator Statement	Processing That Occurs
If (expression, default1, else, default2)	Save the expression and the two possible default values. Include code in the performance test that tests the expression and chooses one of the default values if the value for the parameter is not read from input.
Inout, Input, Output	Set the intent field accordingly.
Extent (expression)	Copy the size along each of the dimensions to the "extent" entry.
NoTouch (condition)	Copy the condition which indicates when this parameter should not be written into the "no touch" entry.

Next, the stub generator generates the stub code for the interface (step 422).

An example of the INPUT, OUTPUT and INOUT code-generator statement follows:

Table 10

```
INTERFACE PARINTENT
        SUBROUTINE SUB(N, M, K, A)
        INTEGER N!#INPUT
        INTEGER M!#OUTPUT
        INTEGER K!#INOUT
        END SUBROUTINE
END INTERFACE
```

When the interface description is processed through the stub generator, the following 32 to 64 bit conversion source is generated:

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

Table 11

```
1    SUBROUTINE SUB (N,M,K)
2    IMPLICIT NONE
3
4    INTEGER*4 N
5    INTEGER(8) :: N_64
6    INTEGER*4M
7    INTEGER(8) ::M_64
8    INTEGER*4K
9    INTEGER(8) ::K_64
10
11   N_64 = N
12
13   K_64 = K
14
15   CALL SUB_64(N_64,M_64,K_64)
16
17   M = M_64
18   K=K_64
19   RETURN
20   END
```

Lines 5, 7, and 9 declare 64-bit integers for the three integer parameters that are passed to the 32 to 64 bit stub. Lines 4, 6 and 8 declare the incoming 32-bit integer parameters. Line 11 copies the incoming value to the 64-bit parameter (N_64) that will be passed to the 64-bit routine since N was indicated to be an intent(input) parameter. Line 13 copies the incoming value to the 64-bit parameter (K_64) that will be passed to the 64-bit routine since K was indicated to be an intent(inout) parameter. Since parameter M was declared intent(output), the incoming value is not copied from the 32-bit to the 64-bit parameter. Line 15 calls the 64-bit routine with the 64-bit parameters. Line 17 copies the value returned by the 64-bit routine (M_64) back to the 32-bit parameter (M) to the caller since M was declared intent(output). Line 18 copies the value returned by the 64-bit routine (K_64) back to the 32-bit parameter (K) to be returned to

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

the caller since K was declared intent(inout). Since parameter N was declared intent(input), the value of the 64-bit parameter (N_64) is not copied back to the 32-bit parameter.

After generating the stub code, the stub code generator determines if there are more subprograms (step 424) and, if so, continues to step (414). Otherwise, processing ends.

Although the present invention has been described with reference to a preferred embodiment thereof, those skilled in the art will know of various changes in form and detail which may be made without departing from the spirit and scope of the present invention as defined in the appended claims and their full scope of equivalents.

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000